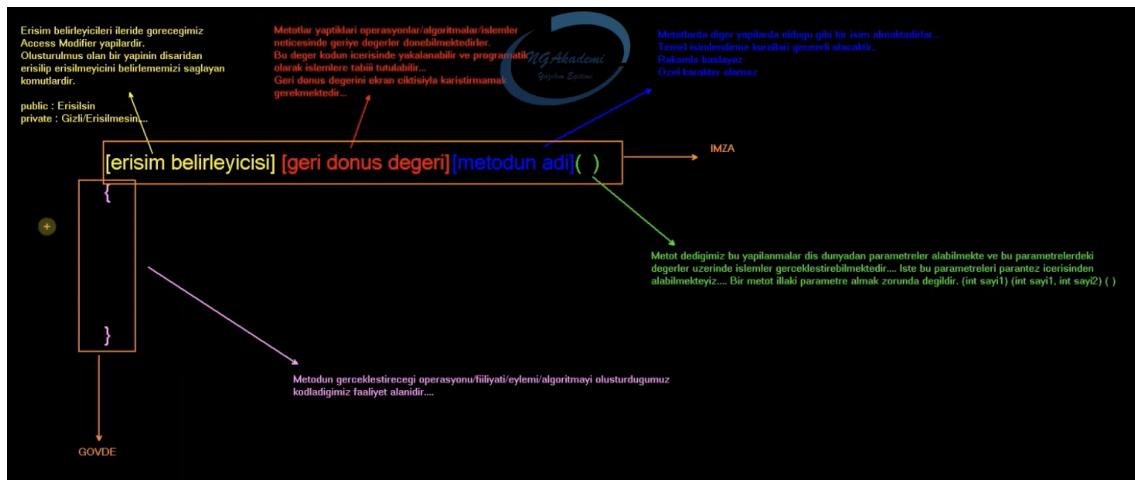


# METOTLAR

## ▼ Nedir?

- Yöntem, yordam, functions
- Yazılımda fiili olarak gerçekleştirdiğimiz tüm operasyonlar bir metottur.
- Prosedürel prog. temel elemanıdır. Olmazsa olmazdır...
- Bir iş, yapan en küçük program parçacıklarıdır.
- Kod tekrarını ortadan kaldırır.
- Kodu tekrar tekrar yazmakla, çağrırmak arasında fark vardır.
- Metotlar bir class içerisinde olmalıdır.
- **Anatomisi:**



- C# da erişim belirleyicisi olarak herhangi bir şey yazmıyorsak private demektir bu. Default özelliği private tir. Java da public tır.

## ▼ İşlevlerine göre metot türleri

- ▼ Geriye değer döndürmeyen, parametre almayan metot

```
//[erişim belirleyici] [geri dönüş değeri] [Metot Adı](.....)
//{

//}

#region Geriye Değer Döndürmeyen, Parametre Almayan Metot

private void |
```

Bir metot geriye değer döndürmüyorsa bunun void ile bildirilmesi ZORUNLUDUDUR!!!

```
private void Metot1 ()
{
}
```

▼ Geriye değer döndürmeyen, parametre alan metot

```
public void Metot2 (int a, bool b, char c)
{
```

▼ Geriye değer döndüren, parametre almayan metot

```
//
//}

Geriye Değer Döndürmeyen, Parametre Almayan Metot
Geriye Değer Döndürmeyen, Parametre Alan Metot
#region Geriye Değer Döndüren, Parametre Almayan Metot

private char Metot5()
{}
```

Eger ki bir metot geriye herhangi bir turde değer döndürecekini ifade ediyorsa kesinlikle o turde bir değer DONDURMELİDIR!!! Aksi takdirde HATA VERİR.

- return keyword ü kullanılır.

```
public char Metot3 ()  
{  
    return 'a';  
}
```



```
0 references  
private int Metot6()  
{  
    if (DateTime.Now.Second > 10)  
        return 5;  
    return 123;  
}
```

•

#### ▼ Geriye değer döndüren, parametre alan metot

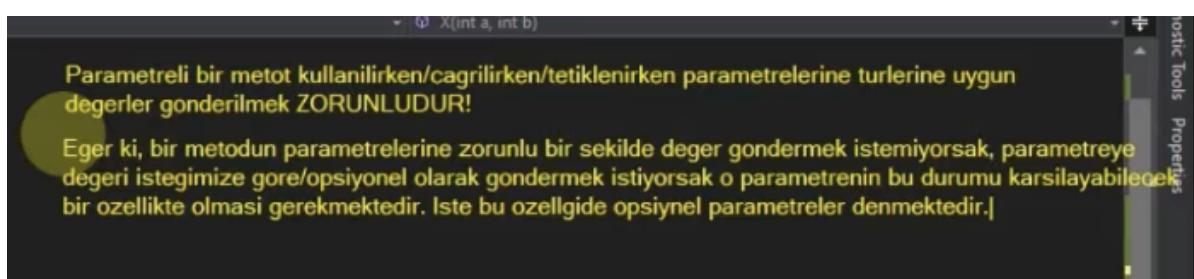
```
private int Metot4 (int a,char b)  
{  
  
    return DateTime.Now.Year > 2000 ? 1:0;  
}
```

## ▼ Metotun geriye değer döndürmesi ne demektir?

- Bir metodun geriye değer döndürmesi demek, içinde yapılan işlem neticesinde üretilen sonuçların ekrana yazılması demek değildir

The screenshot shows a Microsoft Visual Studio interface. On the left is the code editor with a dark theme, displaying a C# program named 'Program.cs'. The code defines a class 'Program' with a static void method 'Main' and a static public int method 'Topla'. A red callout box points to the return statement in 'Topla' with the text: 'Metodun geriye dondurduğu değer, programatik olarak yakalanıp algoritmanın akısında farklı yönlendirmelere sebebiley verebilen değerdir!!!'. Another red callout box points to the 'Main' method with the text: 'Metodun geriye dondurduğu değer, içeriide yapılan işlem neticesinde üretilen değer ekrana/console/veritabanına/herhangi bir log ekranına cıktı vermesi demek DEĞİLDİR!!!'. On the right is the 'Microsoft Visual Studio Debug Console' window, which shows the output of the program's execution. A video feed of a person speaking is visible in the bottom right corner.

## ▼ Mettlarda optional parameters(isteğe bağlı parametreler)



- Bir parametrenin opsiyonel olması demek o parametrenin varsayılan/default değeri olması demektir.

```
namespace functions
{
    class Program
    {
        static void Main(string[] args)
        {
            #region Optional Parameters(İsteğe Bağlı Parametreler)
            X(15, 20);
            #endregion
        }
    }

    static public void X(int a, int b = 0)
    {
    }
}
```

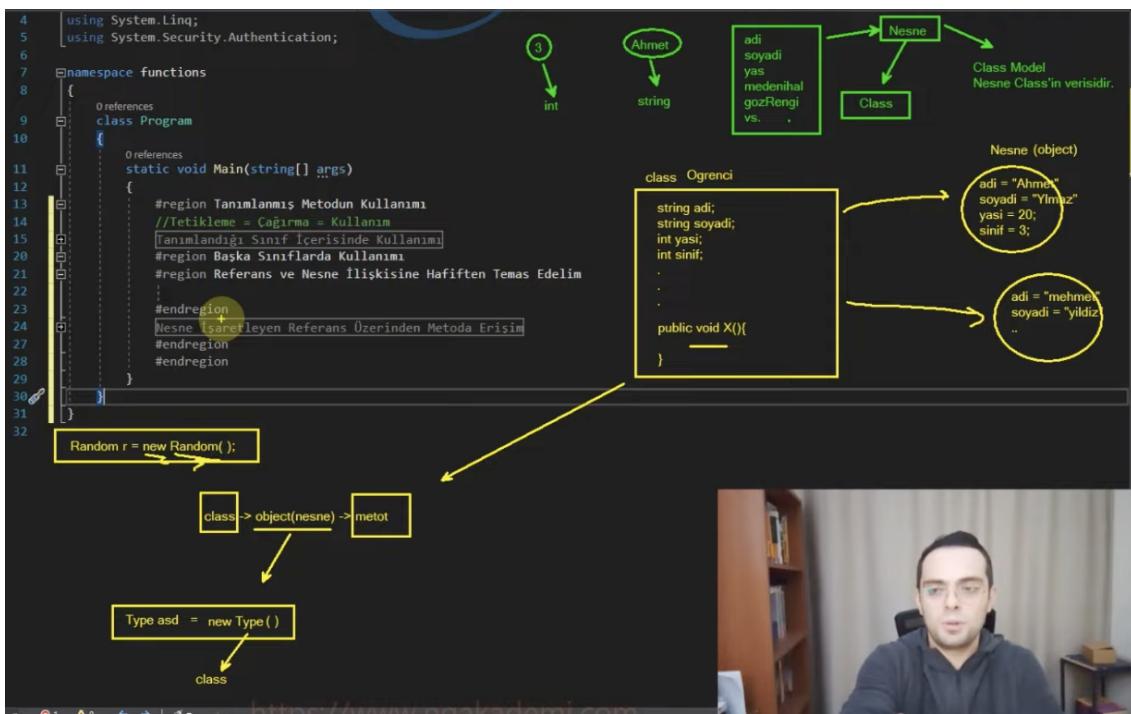
X(5, 7)  
X(5)|

Metot parametrelerine = (assign) operatörü ile bir değer atanırsa o parametreye varsayılan değeri atanmış olur. Haliyle opsiyonel parametre haline getirilmiş olunur...

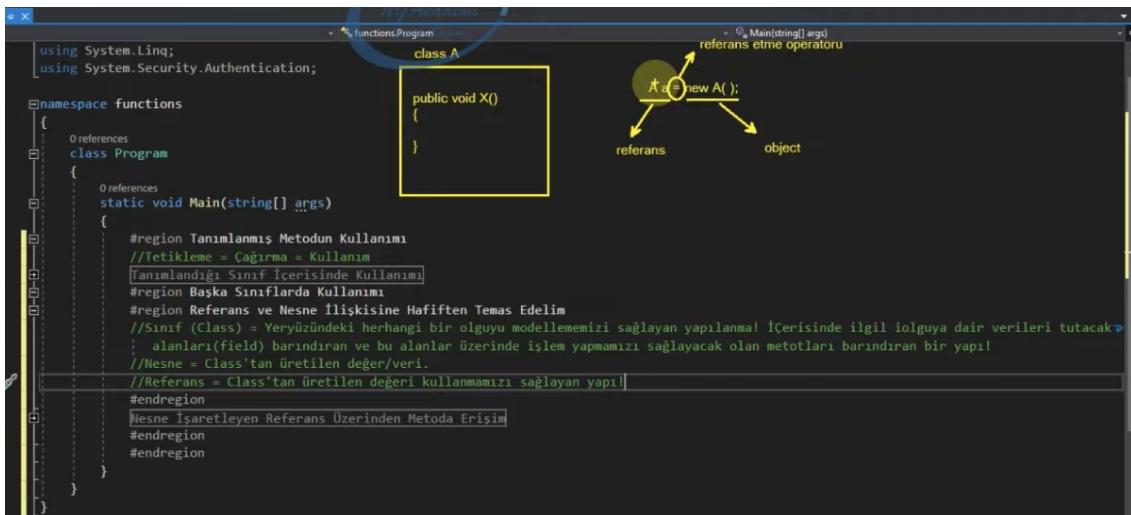
- - ▼ Tanımlanmış metodun kullanımı
  - ▼ Tanımlandığı sınıf içerisinde metodun kullanımı

```
//Bir metod tanımlandığı sınıf içerisindeki farklı bir metod içerisinde çağrılacaksa eğer tek yapılması gereken sadece isminin  
//çagrılmasıdır/tetiklenmesidir/çalıştırılmasıdır.
```

- ▼ Referans ve nesne ilişkisi



- Referans: Oluşturulan nesnenin işaretlenmesini sağlayıp ona erişmemizi sağlayan değişkendir.



### ▼ Başka sınıfta tanımlanmış metodların erişimi

The screenshot shows a code editor window in Visual Studio with the following code:

```
namespace functions
{
    class Program
    {
        static void Main(string[] args)
        {
            #region Başka Sınıfta Tanımlanmış Bir Metodun Kullanımı
            Matematik matematik = new Matematik();
            Console.WriteLine(matematik.Carp(3, 5));
            #endregion
        }
    }

    class Matematik
    {
        public int Topla(int sayi1, int sayi2)
        {
            return sayi1 + sayi2;
        }

        public int Cikar(int sayi1, int sayi2)
        {
            return sayi1 - sayi2;
        }

        private int Carp(int sayi1, int sayi2)
        {
            return sayi1 * sayi2;
        }
    }
}
```

A tooltip is displayed over the `Carp` method call in the `Main` method, listing the following methods from the `Matematik` class:

- Bol
- Cikar
- Equals
- GetHashCode
- GetType
- Topla
- ToString

## ▼ Metotlarda non Trailing Named Arguments Özelliği

- Parametreleri sıra bazlı değil isim bazlı gönderirir. Mümkün olduğunda tüm parametrelerinde bu özelliği kullanırız.
- `X(c:4,a:5,b:"jnj")` şeklinde

## ▼ Metotlarda In Parametreleri (In keywordü)

```

static void Main(string[] args)
{
    //1. parametrenin değerini metodun içerisinde herhangi bir yerde çağrııp kullanabiliriz.
    //2. metod içerisinde üretilen herhangi bir değeri tutacak değişken oluşturmaktaşa parametre üzerinde bu değeri
        tutabiliriz. Yani parametrenin değerini değiştirebiliriz. (Çünkü parametreler özünde bir değişkendir)
    #region In Parameters
    //In komutu sayesinde parametreye verilen değeri sabit tutabilmekteyiz.
    //In komutu, metodun parametresini readonly(Sadece okunabilir) hale getirir.
    //In komutunun kullanılıldığı parametrelerde eğer ki metod içerisinde farklı bir assign durumu söz konusu olursa
        derleyici hatası verecektir.
    #endregion
}
0 references
static void X(in int a)
{
    a = 123;
}
readonly struct System.Int32  

Represents a 32-bit signed integer.

```

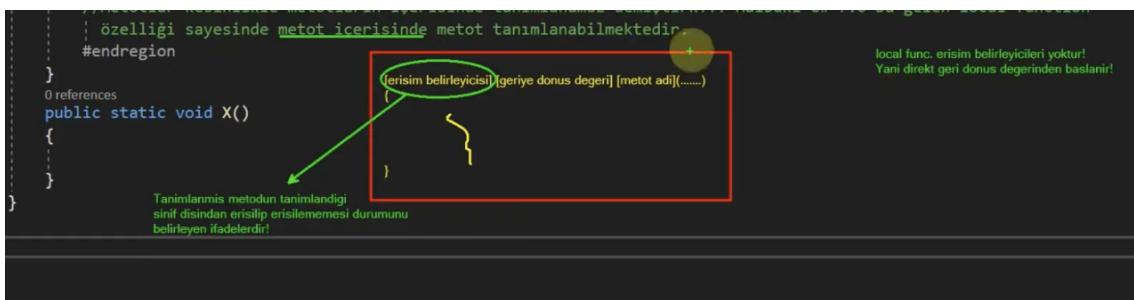
## ▼ Local functions

- Nedir

```

//Bir metod içerisinde tanımlanmış olan metodlardır!
//C#'ta metodlar sade ve sadece class içerisinde tanımlanırlar diye söylemiştık! Halbuki OOP'de göreceğimiz struct
    abstract class, interface, record yapılanmalarında da metodlar tanımlanmaktadır. Metodlar bu saydıklaromızın
    dışında KESİNLİKLE başka bir yerde tanımlanamaz!!!!
//Metodlar kesinlikle metodların içerisinde tanımlanamaz demistir!!! Halbuki C# 7.0'Da gelen local function
    özelliği sayesinde metod içerisinde metod tanımlanabilmektedir.

```



```

    #endregion
    | özellikle sayesinde metod içerisinde metod tanımlanabilmektedir.
    #endregion
}
0 references
public static void X()
{
}

```

local func: erişim belirleyicileri yoktur!  
Yani direkt geri dönüş değerinden başlar!

- Tanımlama kuralları

```

#endif
#region Tanımlama Kuralları
//1. Erişim belirleyici(Access Modifier) yazılmaz!
//2. Local function olarak tanımlanan fonksiyon adı tanımlandığı fonksiyonun adından farklı olmalıdır! Aksi
    taktirde derleyici hatası VERMEZ!!!
#endregion

```

- Kullanım kuralları ve amacı

```
#region Kullanım Kuralları
// Bir local function sadece tanımlandığı metodun içerisinde kullanılabilir.
// Local function tanımlandığı metodun içerisinde her yerden erişilebilir.
#endregion
#region Amacı
// Local function, sadece tek bir metotta tekrarlı bir şekilde kullanılacak bir algoritmayı/kod parçasığını/İşlemi o ↪
| metoda özel bir şekilde tek seferlik tanımlamamızı ve kullanmamızı sağlamaktadır.
#endregion
#region Muadilleri
// Anonim, Delegate, Func
```

- Örnek

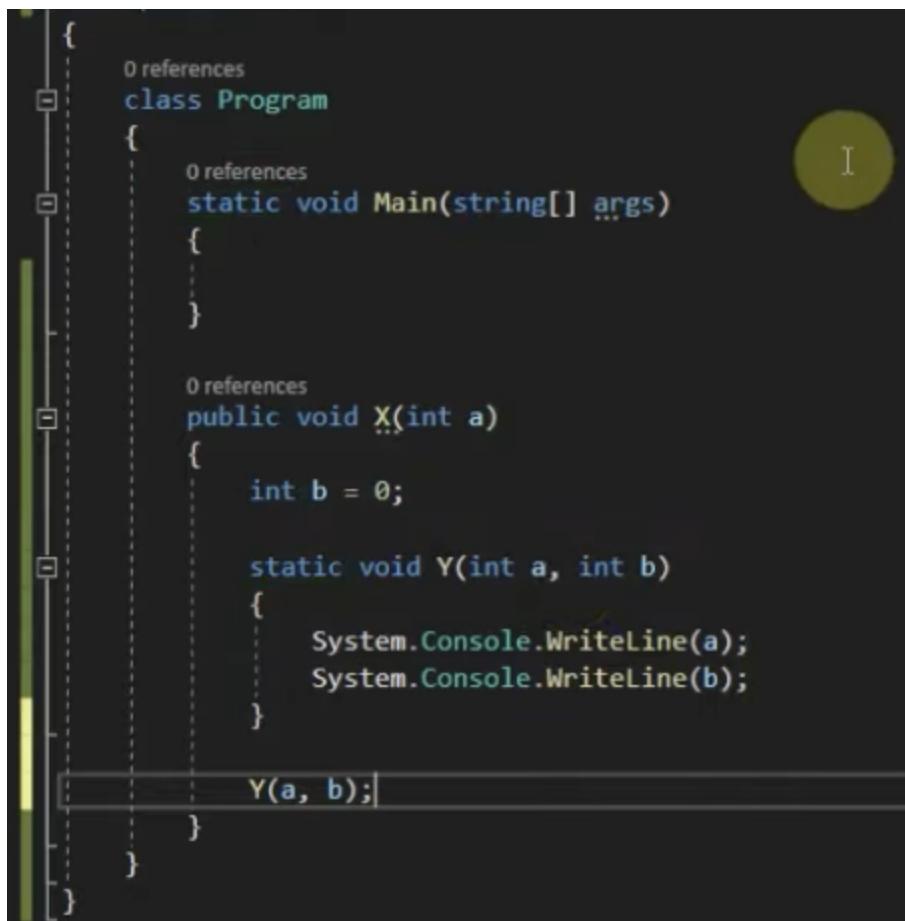
```
public static int X()
{
    Y();

    void Y()
    {
        System.Console.WriteLine("Merhaba");
    }

    Y();

    return 0;
}
```

## ▼ Static Local functions



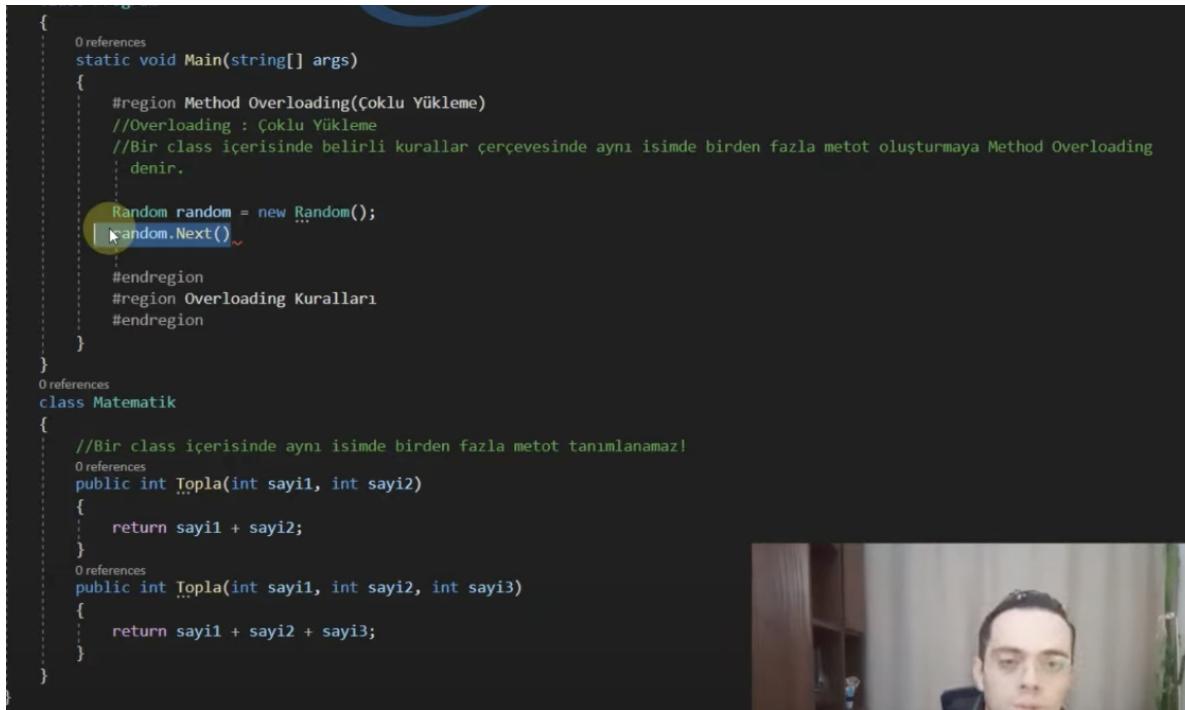
```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        }

    0 references
    public void X(int a)
    {
        int b = 0;

        static void Y(int a, int b)
        {
            System.Console.WriteLine(a);
            System.Console.WriteLine(b);
        }

        Y(a, b);
    }
}
```

- Bunu çok anlamadım?
- ▼ Metotlarda overloading (Çoklu yükleme)



```
0 references
static void Main(string[] args)
{
    #region Method Overloading(Çoklu Yükleme)
    //Overloading : Çoklu Yükleme
    //Bir class içerisinde belirli kurallar çerçevesinde aynı isimde birden fazla metot oluşturmaya Method Overloading denir.

    Random random = new Random();
    Random.Next();

    #endregion
    #region Overloading Kuralları
    #endregion
}

0 references
class Matematik
{
    //Bir class içerisinde aynı isimde birden fazla metot tanımlanamaz!
    public int Topla(int sayı1, int sayı2)
    {
        return sayı1 + sayı2;
    }

    public int Topla(int sayı1, int sayı2, int sayı3)
    {
        return sayı1 + sayı2 + sayı3;
    }
}
```

- Overloading kuralları
  -
- ▼ Recursive (Özyinelemeli / tekrarlamalı) metotlar
- Kendi içerisinde kendini çağırın fonksiyonlardır.
- Öngörülememeyen, derinliği tahmin edilememeyen, sonu bilinmeyen durumlarda tercih edilmektedir.
- Özyinelemeli fonksiyonları sonsuz döngüden parametre ile çıkartabiliriz.
- Aşağıdaki örnekte olduğu gibi.

The screenshot shows a Microsoft Visual Studio IDE window. The main area displays a C# code editor with a file named 'functions.cs'. The code contains a recursive function 'X()' that prints 'Merhaba' to the console. The code is as follows:

```
1 //Recursive Fonk : Kendi içerisinde kendisini çağırın/tetikleyen fonksiyonlardır.
2 //Özyinelemeli/Tekrarlamalı Fonk.
3
4 //Recursive fonk. bir yaklaşımdır!
5
6 //Anlaşılması, kullanımı ve anlatılması zordur!
7
8 //Revursive fonk. ne amaçla kullanılmaktadır?
9 //Öngörülemeyen, derinliği tahmin edilemeyen, sonu bilmeli
10
11 X();
12
13 void X(int a = 1)
14 {
15     Console.WriteLine("Merhaba");
16     if (a < 3)
17     {
18         X(++a);
19     }
20 }
21
22 //void X()
23 //{
24 //    Console.WriteLine("Merhaba");
25 //    X();
26 //    Console.WriteLine("Dünya");
27 //}
```

To the right of the code editor, a yellow-highlighted section of the interface contains the output from the 'Microsoft Visual Studio Debug Console'. The console shows the text 'Merhaba' repeated three times, indicating the recursive call has executed three times.

burada ise 3 defa merhaba, 3 defa dünya yazar.

The screenshot shows a Microsoft Visual Studio IDE window. On the left, the code editor displays a C# file named 'Program.cs' with the following content:

```
1 //Recursive Fonk : Kendi içerisinde kendisini çağırın/tetikleyen fonksiyonlardır.
2 //Özyinelemeli/Tekrarlamalı Fonk.
3
4 //Recursive fonk. bir yaklaşımdır!
5
6 //Anlaşılması, kullanımı ve anlatılması zordur!
7
8 //Reursive fonk. ne amaçla kullanılmaktadır?
9 //Öngörülememen, derinliği tahmin edilemeyen, sonu bilinmeyen d
10
11 X();
12
13 void X(int a = 1)
14 {
15     Console.WriteLine("Merhaba");
16     if (a < 3)
17     {
18         X(++a);
19     }
20     Console.WriteLine("Dünya");
21 }
22
23 void X(int a = 1)
24 {
25     //    Console.WriteLine("Merhaba");
26     //    if (a < 3)
```

A red circular breakpoint marker is visible on line 11. The code editor has syntax highlighting and a vertical margin line. To the right of the editor is a yellow 'Microsoft Visual Studio Debug Console' window showing the following output:

```
Merhaba
Merhaba
Merhaba
Dünya
Dünya
Dünya
```

The console window also contains some text at the bottom: "C:\Users\Gencay\source\repos\functions\fun", "To automatically close the console when de", "bugging stops.", and "Press any key to close this window . . .".

Döngülerin kullanıldığı her yerde recursive kullanabiliriz. Fakat recursive kullanılan yerlerde döngüler kullanılamayabilir.

### ▼ ref keyword'ü

ref nedir?: Referansın kısaltmasıdır.

Referans: OOP kavramıdır. OOP'de Nesneler (Objectler) heap de tutulmaktadır.

Stack ve heap. Heap de tutulanlara bizim direkt erişim yetkimiz yoktur. Bir yazılımcı stacklere direkt erişebilirken, heap e erişemez.

Heap 'de objelerimiz var. Obj: Bir veriden daha fazlasına anlam ifade eder.

Sağım gözüm adım vs vs obje olur.

Heap'e erişebilmek için referans kullanırız.

Referans: Erişebildiğim alanda oluşturabilen ve o alanda heap deki nesneyi referans etmemi sağlayan yapıdır. Yani heapdeki bir objeyi erişilebilir kılan bir değişken türüdür.

Burada bahsettiğim bu davranışı biz değer türlü değişkenlerde nasıl yaparız diye sorarsak; ref keywordü ile yaparız.

Shallow copy kullanmaktadır. Yani veiyi 1 ken 2 yapmıyor. aynısının kullanıyor

```
//Ref keywordü
ref int b = ref a;
//dediğimizde burada ref keywordü ile a nın değerini değil
//referansını b ye iletmiş oluruz.

// Burada, integer türünde b adında bu değişkenimiz artık bi
//edecek bir değişkenken, aynı şekilde a değişkeninin referans
//olarak gönderiyoruz.
//ref a ya a nın değeri değil a nın kendisi gidecektir.
ref int b de a nın kendisini işaretlemiş olacaktır.

//İster a nın değerini değiştir, ister be nin değerini değişt
//hangiisni değiştirirsen değiştir diğerine yansıyacaktır.
```

dediğimizde

local fonksiyon içerisinde recursive fonksiyon kullanırsam

#### ▼ ref return keywordü

sadece metodlarda geçerlidir.

geriye değer döndürebilir.

Geriye nesneler ve referanslar da döndürebiliriz. Ayrıca değer türlü değişkenlerin referanslarını da geriye döndürebiliriz.

#### ▼ out keywordü